(4)

Decomposition of Linear Programs Using Parallel Computation*

James K. Ho
Tak C. Lee
R.P. Sundarraj

Management Science Program
College of Business Administration
University of Tennessee
Knoxville, TN 37996-0562

Revised: May, 1988

To appear in *Mathematical Programming*

*Invited paper at the Symposium on Parallel Optimization, Madison WI, August 10-12, 1987.
Research supported in part by the Office of Naval Research under grant N00014-87-K-0163.

(A)

Abstract

This paper describes DECOMPAR: an implementation of the Dantzig-Wolfe decomposition algorithm for block-angular linear programs using parallel processing of the subproblems. The software is based on a robust experimental code for LP decomposition and runs on the CRYSTAL multicomputer at the University of Wisconsin-Madison. Initial computational experience is reported. Promising directions in future development of this approach are discussed. (KR) ←

Keywords: Linear Programming, Large-Scale Systems, Decomposition, Parallel Computing.

1. Introduction

Many linear programming models represent large, complex systems consisting of independent subsystems coupled by global constraints. Such LP's are said to have the block-angular structure. The decomposition principle of Dantzig and Wolfe [1] leads to algorithms that transform the original problem into a sequence of subproblems corresponding to the uncoupled subsystems. The subproblems are coordinated by a master problem corresponding to the global constraints through primal (proposals) and dual (prices) information. While it has been obvious that the subproblems can be solved simultaneously for algorithmic efficiency, it is not until recently that advances in computer technology make such an approach realizable.

Advances in VLSI (very large-scale integration) for digital circuit design are leading to much less expensive and much smaller computers. They have also made it possible to build a variety of "supercomputers" consisting of many small computers combined into an array of concurrent processors. We shall refer to such an architecture as multicomputers. Each individual processor is called a node. Visionaries in the industry are discussing designs with 10^4 to 10^5 nodes each capable of 10 megaflops. Actually, machines with 10^2 to 10^4 nodes of varying power are already available commerically. They cost well over $\$10^5$ currently. However, realistic projections call for prices to be lowered by at least a factor of 10 within five to ten years.

The prospect of LP decomposition using parallel computation has significant impact on many real-world applications. We shall cite only one example for illustration. An important problem in production and operations management is that of material requirements planning (MRP). A number of products are to be assembled from parts which are themselves made up of other parts. Each step in the assembly requires certain manufacturing capacities. The exogenous demands for the products and spare parts in each time period over a finite planning horizon are projected. Assuming linear production and inventory holding costs, the problem is to find the least cost production and inventory schedule that stays within the capacity constraints and meets the demands. Formulated as an LP, a 10-period problem with 100 products, each with 100 parts, will have on the order of 100,000 constraints and

200,000 variables. Previously, such problems were simply too large for existing LP software and the LP approach was deemed impracticable. In [3], Ho and McKenney showed that using decomposition, the subproblems have the property that any basis can be triangularized. This allows network-like techniques to be implemented on nodes of existing multicomputers to handle the subproblems. Using a multicomputer with 100 nodes, the LP decomposition approach is expected to be viable.

Before such specialized planning systems can be realized, empirical studies of LP decomposition using parallel computation will be important first steps. This paper describes DECOMPAR: an implementation of the Dantzig-Wolfe decomposition algorithm for block-angular LP's on a multicomputer. Section 2 reviews the basic algorithm and summarizes recent developments. The essential features of an efficient implementation on conventional computers are described in Section 3. Certain aspects of CRYSTAL, an experimental multicomputer at the University of Wisconsin-Madison, relevant to LP decomposition are discussed in Section 4. Section 5 presents the design of DECOMPAR. Initial computational results are reported in Section 6 and discussed in Section 7.

2. Review of Dantzig-Wolfe Decomposition

For brevity, the algorithmic details will be omitted here. The reader is referred to Ho and Loute [5] for a concise summary of the method. As a quick review, however, it should be helpful to capture the essential ideas graphically. The structure of the block-angular LP is depicted in Figure 1. The master problem is illustrated in Figure 2 where X_r indicates an array of extreme point solutions to the r^{th} subproblem. A column in the master problem is known as a proposal from a subproblem. A typical subproblem is shown in Figure 3. Note that in general extreme ray solutions to the Subproblems must also be considered. The master problem determines an optimal combination of the proposals on hand by assigning values to the weights λ. The optimal dual variables π, known as prices, are used to adjust the objective function in the subproblems which in turn may produce new proposals to improve the global objective in the master problem. The process continues until no further improvement can be achieved.

A discussion of historical perspective as well as recent developments in the LP decomposition approach is given in Ho [4]. The main points of that survey are summarized below.

i) Due to the lack of extensive and systematic studies of the behavior of decomposition algorithms, early experiences may have left a generally negative and misleading imprint on the entire approach.

ii) For well-behaved LP models, the convergence as measured by the number of times the master problems must be resolved is actually surprisingly fast.

iii) Slow convergence may be caused, at least in part, by the propagation of numerical errors.

iv) There are indeed meaningful, large-scale applications that eventually may have to rely on decomposition.

v) New theoretical results are leading to more efficient and robust algorithms.

vi) New computer architectures allowing parallel computation will provide further opportunities to realize the potential of the decomposition approach.

3. DECOMP: A Sequential Decomposition Code

As the building blocks of our parallel implementation, we use DECOMP: a Fortran code of the Dantzig-Wolfe decomposition algorithm. This code was first assembled in 1973 by Carlos Winkler at the Systems Optimization Laboratory at Stanford University. It was based on John Tomlin's LPM1 code of the revised simplex method. Since then, DECOMP has been extended and improved over a number of years by James Ho and Etienne Loute at the Center for Operations Research and Econometrics in Belgium. It was used extensively in empirical studies (see e.g. [6]) and as prototype for more advanced, commercial software based implementations (e.g. DECOMPSX in [5]). Comprehensive documentation for DECOMP was completed recently by R.P. Sundarraj [7].

While many refinements were necessary to make DECOMP comput- ationally efficient and robust, we list only the major ones. Since all of these techniques are also used in the advanced implementation DECOMPSX described in [5], the reader

may refer to that paper for further details.

i) Data is disk resident. The LP to be solved (either the master problem or a subproblem) is read into memory. Subsequent modifications are written to disk.

ii) All matrices are stored in sparse form using column packing. The coupling coefficients (A_r) are included in a subproblem as nonbinding constraints (Figure 4) whose updated right-hand-side gives a proposal to the master problem.

iii) The prices π are incorporated directly into the dual variables of the subproblems without actual modification of its objective function.

4. CRYSTAL: A Multicomputer

For research on LP decomposition using parallel computation, we are primarily interested in multiple-instruction-multiple-data (MIMD) multicomputers. These may be integrated systems with multiple processors, local-area-networks (LAN) of mini- or microcomputers, or even networks of mainframes. We are not concerned with parallel computing in the sense of SIMD, pipeline or vector machines.

The CRYSTAL multicomputer [2] under development in the Computer Science Department at the University of Wisconsin-Madison provides a distributed processing environment most suitable for experimentation with LP decomposition. It is a set of 20 VAX-11/750 minicomputers each with 2 megabytes of memory connected by a 80 megabit/sec Proteon ProNet token ring. Depending on the amount of direct control of processor resources required, projects can be implemented using either a communication service (the "nugget") that resides on each node processor, or the Charlotte distributed operating system. Development, debugging and execution of projects are done through any of several VAX-11/780 hosts running under Berkeley Unix 4.2. The CRYSTAL project is supported by a National Science Foundation Coordinated Experimental Research grant. Access to the systems for the present work is provided by courtesy of Professor Robert

Meyer.

Since LP decomposition has a natural interpretation as decentralized planning, the granularity (or loose coupling) of the CRYSTAL architecture is of particular relevance. The nodes, being separate machines, can in principle represent geographically distinct facilities. This will allow, for example, corporate-level LP modeling with divisions handling their own subproblems. Or, in the case of multinational systems analysis, individual nations may guard their sensitive data and still be able to contribute to a joint project. For this reason, the speed-up obtainable with parallel computing is not the only motive behind our empirical studies. The efficacy of distributed processing is also of paramount interest.

Several aspects of the CRYSTAL environment that affect our experiments should be mentioned here. First, the host machines operate under time-sharing. Therefore run-times depend on the machine load. Secondly, more than one host may be active and message speed on the token ring is again load dependent. Finally, it is possible for errors to occur in messages. The token ring monitors errors automatically and retransmits messages if necessary. These factors can introduce considerable variability in experimental results. In Section 6, we discuss the measures of performance used in our experiments that remain significant under such circumstances.

5. DECOMPAR: A Parallel Decomposition Code

The basic idea behind the design of DECOMPAR is to process the subproblems concurrently on the node machines. In the current version, each subproblem is assigned to one node and the master problem is handled by the host. A schematic representation of DECOMPAR is shown in Figure 5. To allow for situations where there are more subproblems than nodes, later versions will have built-in mechanisms to distribute the subproblems appropriately. Also, since time-sharing on the host machine makes it difficult to obtain accurate timing of algorithmic procedures, the option of having the master problem on a node will also be implemented.

The procedures used on the host machine are listed in Table 1 together with the functions they perform. Those for the node machines are listed in Table 2.

Type	Name	Function
I/O & Setup	INDATA	Driver for input routines
	INIT	Initialize communication buffers
	INPUT	Read data in MPS format
	SENDAT	Distribute subproblems to nodes
	UNRAVL	Write solutions
Decomposition	MASTER	Process the master problem
	PACK	Incorporate proposals into master
	POLICY	Set parameters according to strategy
	RESULT	Phase 3 of decomposition to reconstruct a primal solution to LP
Revised Simplex	BTRAN	Backward tranformation for prices
	CHSOL	Check accuracy of solution
	CHUZR	Ratio tests for column to leave basis
	FORMC	Check feasibility
	FTRAN	Forward transformation for updated column
	INVERT	Refactorization of the basis
	NORMAL	Primal Revised Simplex
	PRICE	Pricing for column to enter basis
	UNPACK	Unpack a column in the LP matrix
	UPBETA	Update the right-hand-side
	WRETA	Update the basis factorization

Table 1. DECOMPAR Procedures on the Host Machine

Type	Name	Function
Decomposition	SUB	Driver to process subproblem
	CHECK	Regulate proposal generation
Revised Simplex	(Same procedures as on the Host machine)	

Table 2. DECOMPAR Procedures on the Node Machines.

The following is a pseudo-code description of DECOMPAR in what will be referred to as the standard version. Variations of this scheme will be implemented to accomodate different strategies made possible by parallel computation.

DECOMPAR: Host Program

Step 1: Input data

1.1 Read and store master problem data.

1.2 For each subproblem,

1.2.1 read data;

1.2.2 send data to node;

1.2.3 wait to receive proposal from node.

1.3 Incorporate proposals into master problem.

Step 2: Iterations

2.1 Phase 1

2.1.1 Set Phase 1 objective in master problem.

2.1.2 CYCLE until termination.

2.1.3 If infeasible, stop;

else 2.2.

2.2 Phase 2

2.2.1 Set Phase 2 objective in master problem.

2.2.2 CYCLE until termination.

2.2.3 If unbounded, stop;

else Step 3.

CYCLE:

C.1 Solve master problem.

C.2 If (primal-dual gap < tolerance) or (no more proposals), terminate.

C.3 Send prices to nodes.

C.4 Wait to receive proposals from all nodes.

C.5 Incorporate proposals into master problem.

C.6 Return to C.1.

Step 3: Phase 3

3.1 Compute allocations for subproblems.

3.2 Send allocations to nodes.

3.3 Output solutions from nodes.

DECOMPAR: Node Program

Step 1: Initialization

1.1 Wait to receive subproblem data from host.

1.2 Solve subproblem.

1.3 If infeasible, stop;
else generate proposal.

1.4 Send proposal to host.

Step 2: Iteration

2.1 Wait for prices from host.

2.2 Solve subproblem.

2.3 Generate proposals, if any.

2.4 Send proposals to host.

2.5 Return to 2.1.

The load-balancing aspect of the standard version of DECOMPAR is illustrated in Figure 6. There, a shaded time slot indicates a busy period for the corresponding machine. A blank indicates an idle period. The time for a cycle consists of that required to process the master problem on the host plus the longest time to process a subproblem on the nodes. These critical times sum up to the total solution time and are drawn in darker shades.

Currently, DECOMPAR is dimensioned for block-angular LP's with up to 10 subproblems, each with up to 400 rows, 1000 columns and 10,000 nonzeros for matrix and basis data. The master problem can have up to 99 rows (including convexity constraints). While all the dimensions can be expanded considerably within the CRYSTAL environment, the only maior adjustment planned for initial experiments is to increase the number of subproblems and the number of coupling rows.

From Figure 6, it is quite clear that load balancing can be improved by using variations of the standard strategy aimed to keep the machines busy more of the time. In particular, we consider the following strategies:

I. *First Subproblem Strategy*:
Master is activated as soon as first subproblem with proposal is completed.

II. *First Proposal Strategy*:
Master is activated as soon as first proposal is generated.

III. *Instant Feedback Strategy*:
All nodes are kept active if possible with prices and proposals communicated as soon as available.

While there should be less idle time using these strategies, the speed of convergence may be adversely affected. This is because the information that is being generated and communicated more rapidly may require many smaller steps toward the final solution. Such algorithmic behavior will obviously be problem dependent and can be better understood only through empirical observations. The various

options available in the DECOMPAR code are therefore useful for the investigation of parallel computation in LP decomposition.

6. Computational Results

The initial experiments are designed mainly to validate DECOMPAR and briefly compare the computational strategies. Ten small to medium size test problems are used. Their characteristics are listed in Table 3.

Problem	Blocks		Rows		Columns	% Density
	Actual	Natural	Coupling	Total		
DEEP1	6	6	16	100	264	29.3
DEEP2	4	4	11	100	225	32.8
FIXMAR	4	4	18	325	777	1.2
MEAT12	6	12	46	381	692	1.3
MEAT31	8	31	11	384	961	1.3
MEAT43	9	43	16	648	1253	0.7
FORESTRY	6	6	11	402	1006	1.0
SCORPION	6	6	53	389	747	0.7
DYNAMICOa	5	10	10	678	1177	0.7
DYNAMICO	10	10	10	678	1177	0.7

Table 3. Characteristics of the Test Problems

DEEP1 and DEEP2 are randomly generated problems with dense blocks. All the others are from real applications. FIXMAR is a production scheduling problem. The MEATxx examples are for multiproduct ingredient mix optimization in the meat processing industry. FORESTRY is from a forest management model.

SCORPION is from a French energy model. DYNAMICO is a model of world trade and development from the United Nations. The column counts in Table 3 include one logical column per row. For some problems, several natural blocks are grouped together to reduce the actual number of subproblems required. The termination criterion used throughout is a relative tolerance of 10^{-4} for the primal-dual gap.

Since the host machine is time-shared, every effort is made to run comparative experiments under close to identical conditions. While these initial results are not intented to provide definitive comparative measures, they do give an idea of the relative performance of various options in DECOMPAR and suggest directions for future improvement. The total run time with DECOMPAR is compared to that with DECOMP on the host machine. For DECOMP, the total includes disk I/O time which is an essential feature of sequential decomposition because the subproblems have to be disk resident. The approximate percentage of the total time involved in disk I/O is also recorded. The speed-up factor is the ratio of total DECOMP time to total DECOMPAR time. Note that the total DECOMPAR time includes message transmission times on the token ring. At present, we do not have accurate measures of such intermodal communications. The estimate is that they constitute less than 1% of the total DECOMPAR time. The run times and speed-up factors for the standard version of DECOMPAR are recorded in Table 4. All times reported exclude the initial input times for problem data.

Problem	Cycles	DECOMP		DECOMPAR	Speed-Up
		Total	%I/O		
DEEP1	14	393	63	106	3.72
DEEP2	14	275	71	58.0	4.75
FIXMAR	19	491	54	128	3.83
MEAT12	16	582	49	238	2.45
MEAT31	7	108	61	15.5	6.94
MEAT43	8	128	72	16.4	7.80
FORESTRY	14	506	44	138	3.67
SCORPION	5	63.0	76	8.51	7.41
DYNAMICOa	17	1004	39	192	5.23
DYNAMICO	11	530	46	97.8	5.42

Table 4. Solution Times (in seconds) for DECOMP and DECOMPAR (standard version).

Two standard measures of effectiveness in parallel computing are efficiency and load-balance. Efficiency is expressed as $E = S / P$ where S is the speed-up factor and P is the number of processors used. In most applications, $E_{max} = 1$ is the theoretical limit on efficiency. Load-balance is indicated by the busy time as a percentage of total time on the processors. Perfect load-balance is when all the processors are 100% utilized. In general, applications with higher efficiency and better load-balance are considered more effective in parallel computing.

By nature of the decomposition codes, the theoretical limit of E is not 1. This is because we are not simply distributing a fixed amount of work over several processors. Instead, we assume that a single processor cannot in general solve an LP all-in-core using decomposition. With DECOMP, the subproblems are disk resident and considerable disk I/O is incurred in setting them up for solution sequentially.

With DECOMPAR, all data associated with a subproblem is stored in the local memory of a node machine and no disk I/O is necessary. Of course, the master and subproblems have to communicate by passing messages. It turns out that by comparison, the amount of such data transfer required is much less than the manipulation of subproblems. Therefore, DECOMPAR is actually doing *less* total work than DECOMP when solving the same problem and the efficiency of parallel computation can exceed one. It should be remarked that customarily, only CPU times are compared in computational experiments. However, for large scale problems such as linear programs for production and operations planning, the wall-clock time is of ultimate practical interest. In this respect, the potential benefits of LP decomposition with parallel computation are particularly promising.

Note that since we are really comparing different tasks, the actual speed-up factors will depend on the relative efficiency of computation and disk I/O. For example, they will be lower if a faster disk drive is used. For this reason, our results comparing sequential and parallel decomposition serve mainly as a case for reference.

In Table 5, the speed-up factors for DECOMPAR are used to compute E. The percentage utilization of the host and the average percentage utilization of the nodes are also recorded. Observe that the major factor in the speed-ups obtained with DECOMPAR as compared to DECOMP is the saving in disk I/O time. In terms of computation, the standard version of DECOMPAR is essentially a two-tier algorithm because each cycle consists of a master problem and then the subproblems. Only the latter are processed in parallel. Suppose a problem with r blocks takes t cycles in which the average time for the master is m and the average time for a subproblem is s. Then the total sequential time is approximately $t(m + rs)$ disregarding time to manipulate data. The total parallel time using one processor for each of the master and subproblems is approximately $t(m + s)$. Therefore, letting $P = r + 1$, we have

$$E_{max} \approx (m + rs)/(m + s)(r + 1).$$

For m close to s, $E_{max} \approx 1/2$.

Problem	P	% Host	% Nodes	Speed-Up	E
DEEP2	5	58.7	21.1	4.75	0.95
FIXMAR	5	61.6	18.5	3.83	0.77
DYNAMICOa	6	56.7	27.9	5.23	0.87
DEEP1	7	77.9	10.9	3.72	0.53
MEAT12	7	92.5	4.28	2.45	0.35
FORESTRY	7	42.8	25.6	3.67	0.52
SCORPION	7	84.7	8.67	7.41	1.06
MEAT31	9	64.3	5.78	6.94	0.77
MEAT43	10	78.7	8.48	7.80	0.78
DYNAMICO	11	53.3	24.7	5.42	0.49
MEAN	7.4	67.1	15.6	5.12	0.71

Table 5. Efficiency and Load-Balance Measures for DECOMPAR (standard version)

All the options implemented in DECOMPAR (first subproblem strategy, first proposal strategy and a variant of instant feedback strategy) are aimed to improve the load balancing of the decomposition approach. Empirical results with these options are summarized below.

From Table 6 it can be seen that while the strategy of restarting the master problem as soon as the first subproblem terminates with a proposal does increase the utilization of the processors, the total run time is significantly inferior to the standard version in most cases tested. This is because such earlier restarts often preclude the generation of useful information in a particular cycle, hence increasing the number of cycles required and offsetting any benefits from an improved load balance.

Problem	Cycles	% Host	% Nodes	Speed-Up (relative to standard version)
DEEP2	32	68.2	31.6	0.86
FIXMAR	102	79.4	30.2	0.59
DEEP1	43	86.8	13.0	0.68
MEAT12	57	92.3	7.6	1.05
FORESTRY	61	64.8	34.9	0.85
SCORPION	19	88.2	11.4	0.42

Table 6. Performance of DECOMPAR with First Subproblem Strategy

These results implied that the second strategy of restarting the master problem as soon as a proposal is generated will be even less interesting. For this reason, we did not make any runs using the first proposal strategy. At this point, the only remaining option of improving on the standard version is instant feedback, i.e. passing prices and proposal information among the master and subproblems as soon as they are generated. In principle, the purest form of this strategy can be implemented as follows. At every simplex step in the master problem, the new prices are sent to the subproblems in appropriate buffers. At every simplex step in a subproblem, the latest master prices in the buffer is incorporated into the complete pricing vector which has the same effect as modifying the objective function of the subproblem. A proposal is sent at once to the master problem and will be considered in the next simplex pricing step there. It turns out that for practical problems, this scheme will very quickly create an unmanageable amount of intermodal communication among the master and subproblems. Modifications are therefore necessary to regulate such coordinating information.

We shall call the version of modified instant feedback strategy that has been

tested on CRYSTAL the accelerated feedback strategy. When the master problem is idle, it keeps checking the appropriate buffer for proposals. As soon as one or more proposals are found, they are incorporated and the master problem is solved. Meanwhile, the subproblems may continue to generate proposals according to the last set of prices from the master problem. These proposals are put in the buffer. At optimality of the master problem, prices are sent to the subproblems to replace the old ones immediately. The master problem again checks for proposals in the buffer and restarts if there is any. This represents a better regulated update of price information than instant feedback. Instead of updating at every simplex step of the master problem, new prices are typically sent to the subproblems after several simplex steps. The process between two successive price updates constitutes a cycle. As to the proposals, experience with DECOMP established that for best results, some selection mechanism is necessary when sending multiple proposals from a subproblem. This is because an excessive number of proposals may clutter up the master problem. Also, proposals that are too similar may introduce numerical instability in the master problem. Therefore, as admissible proposals are being generated, a screening mechanism regulated by user-supplied parameters is used.

Offhand, it is not obvious that overall performance can be improved. While we attempt to keep all the processors busy, they may be generating inferior information that leads to poor convergence. As it turns out, this strategy works very well and in all cases tested, succeeds in increasing both the processors utilization and the overall speed-up. On the average, the accelerated feedback strategy is 1.31 times faster than the standard version. The host utilization increased from 67.1% to 95.8%, the nodes utilization from 15.6% to 21.0%. The results are summarized in Table 7. Note that even with accelerated feedback, the average utilization of the subproblem nodes is still quite low. This shows that the subproblems are relatively "easy", especially in later stages of the solution process. An obvious strategy to take advantage of this fact is to assign more than one subproblem to a node. However, this will involve substantial modification of the codes and is therefore not actually tested.

Problem	Cycles	% Host	% Nodes	Speed-Up (relative to standard version)
DEEP2	24	94.0	31.8	1.38
FIXMAR	52	98.1	25.8	1.16
DYNAMICOa	49	93.6	42.2	1.41
DEEP1	29	98.7	12.5	0.88
MEAT12	38	98.4	5.43	1.02
FORESTRY	36	96.2	34.8	1.74
SCORPION	10	91.1	15.4	1.24
MEAT31	15	96.9	8.25	1.04
MEAT43	12	94.9	12.7	1.93
MEAN	29	95.8	21.0	1.31

Table 7. Performance of DECOMPAR with Accelerated Feedback Strategy

7. Discussion

We have demonstrated DECOMPAR as a robust experimental tool for LP decomposition using parallel computation. It is now possible to investigate many interesting computational strategies. Our initial empirical results indicate that significant speed-up can be obtained using parallel decomposition mainly because of the elimination of the disk I/O required in sequential decomposition to manipulate the subproblems. To further streamline the computations, the accelerated feedback strategy is very promising. Apart from algorithmic efficiency, this observation may have significant implications in the analysis of information flow within decision

processes. For our test problems, it is observed that the processor utilization for the master problem dominates that for the subproblems. This suggests that each node should be used to handle several subproblems. Also, more sophisticated controls can be designed to allocate the work load dynamically. Our plans are to investigate such topics using commercially available multicomputers of both distributed and shared-memory architures. Test problems from material requirement planning with around ten thousand constraints will be used. Further work in this direction should contribute to the goal of solving large-scale production and operations planning problems on affordable parallel computers.

Acknowledgement

The authors are indebted to Robert Meyer for providing access to the CRYSTAL system. They also wish to thank R. J. Chen, Deepankar Medhi, Russell Lee, and Charles Pfleeger for their help in overcoming various obstacles in the course of this project.

References

[1] G.B. Dantzig and P. Wolfe, "The decomposition principle for linear programs", *Operations Research* 8 (1960) 101-111.

[2] D. Dewitt, R. Finkel and M. Solomon, "The CRYSTAL multicomputer: design and implementation experience", Technical Report 553, Computer Science Department, The University of Wisconsin-Madison, September, 1984.

[3] J.K. Ho and W. McKenney, "Triangularity of the basis in linear programs for material requirements planning" Technical Repert MSP-87-3, Management Science Program, The University of Tennessee, Knoxville, July 1987.

[4] J.K. Ho, "Recent advances in the decomposition approach to linear programming", Mathematical Programming Study 31 (1987) 119-128.

[5] J.K. Ho and E. Loute, "An advanced implementation of the Dantzig-Wolfe decomposition algorithm for linear programming", *Mathematical Programming* 20 (1981) 303-326.

[6] J.K. Ho and E. Loute, "Computational aspects of DYNAMICO: a model of trade and development in the world economy", *Revue Française d'Automatique, Informatique et Recherche Opérationelle* 18 (1984) 403-414.

[7] R.P. Sundarraj, "Documentation of DECOMP: a Dantzig-Wolfe decomposition code for linear programming", Master's Thesis, Management Science Program, The University of Tennessee, Knoxville, 1987.

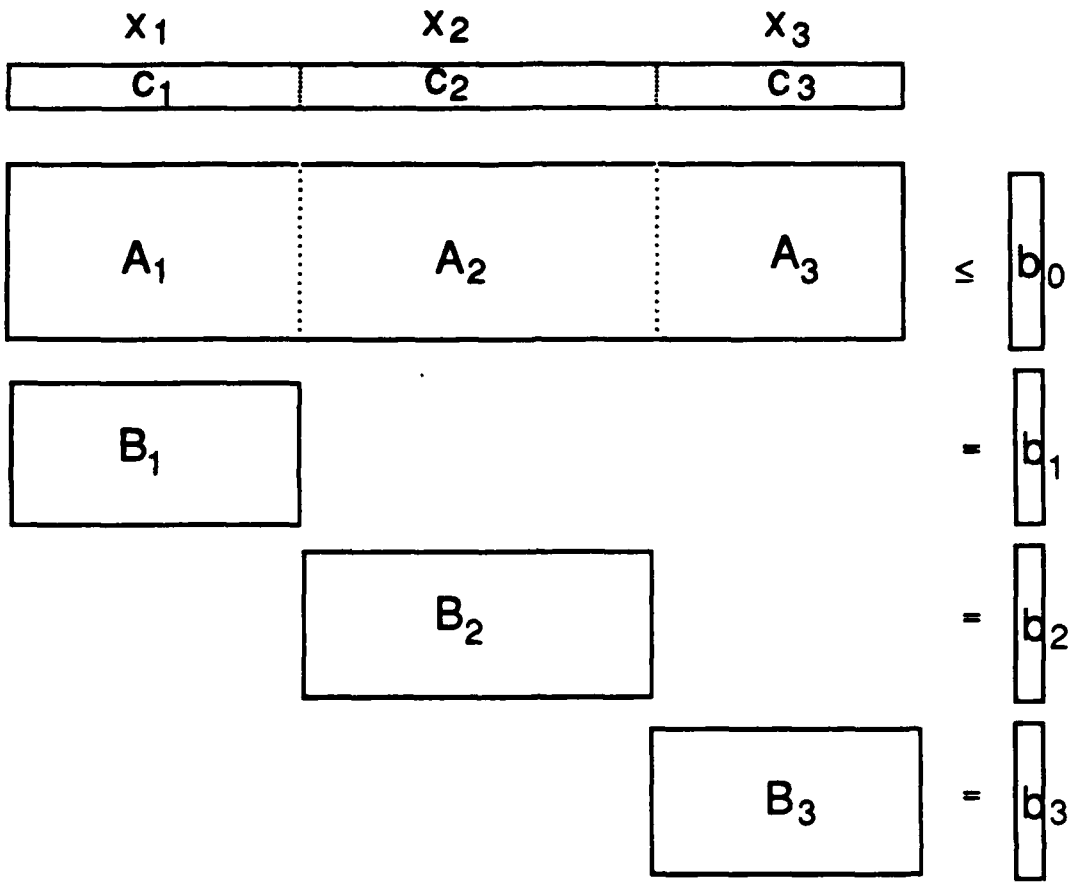

Figure 1. A Block-Angular Linear Program

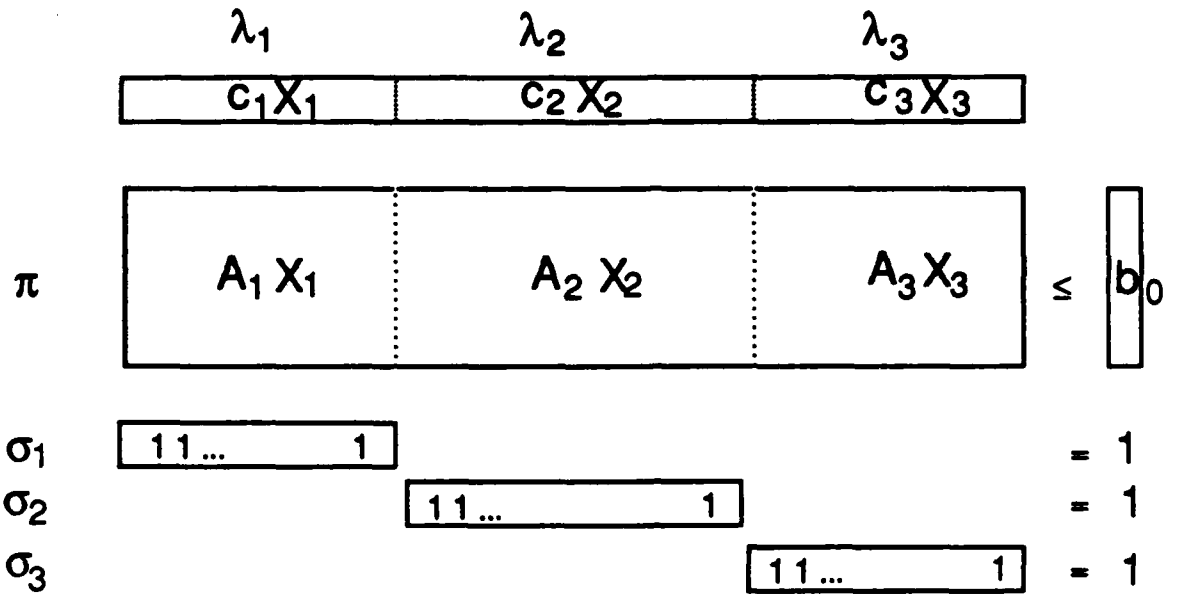

Figure 2 The Master Problem

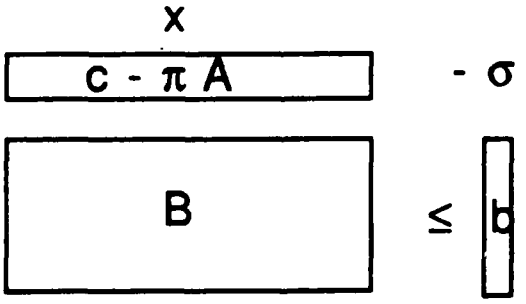

Figure 3 A Subproblem

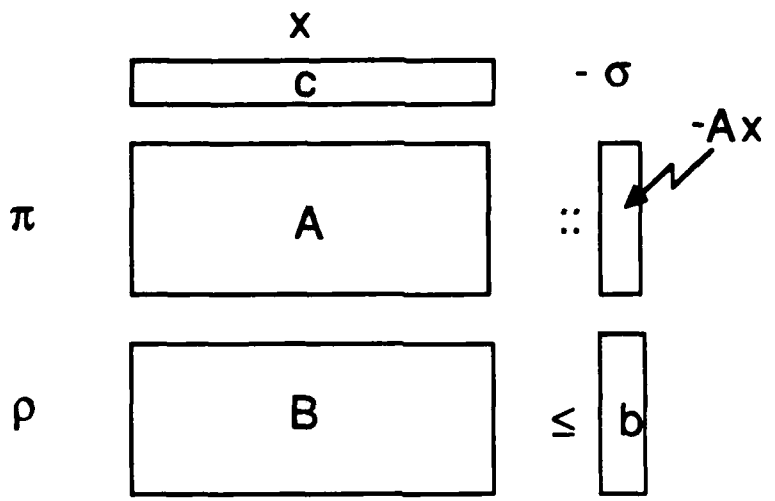

Figure 4 A Subproblem as implemented

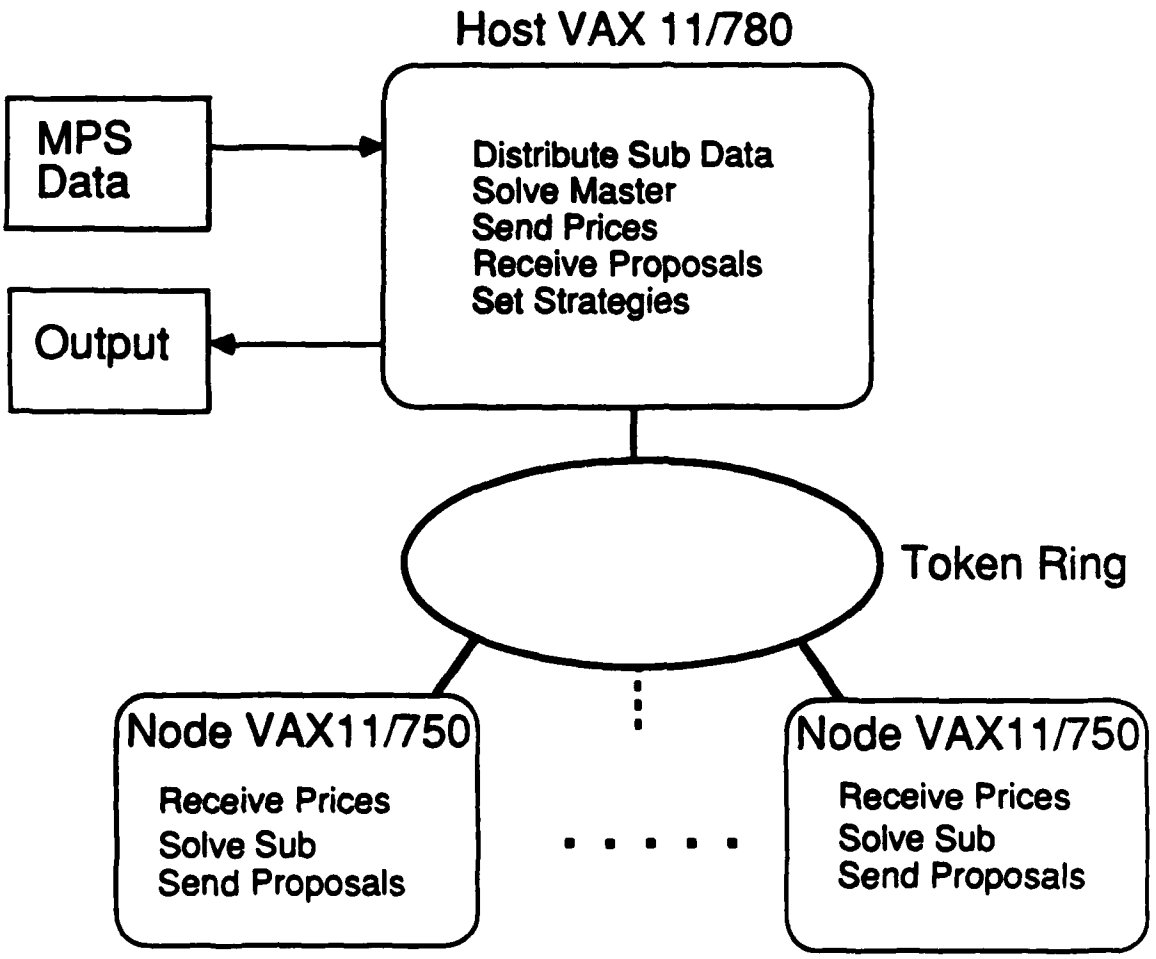

Figure 5 Design of DECOMPAR on CRYSTAL

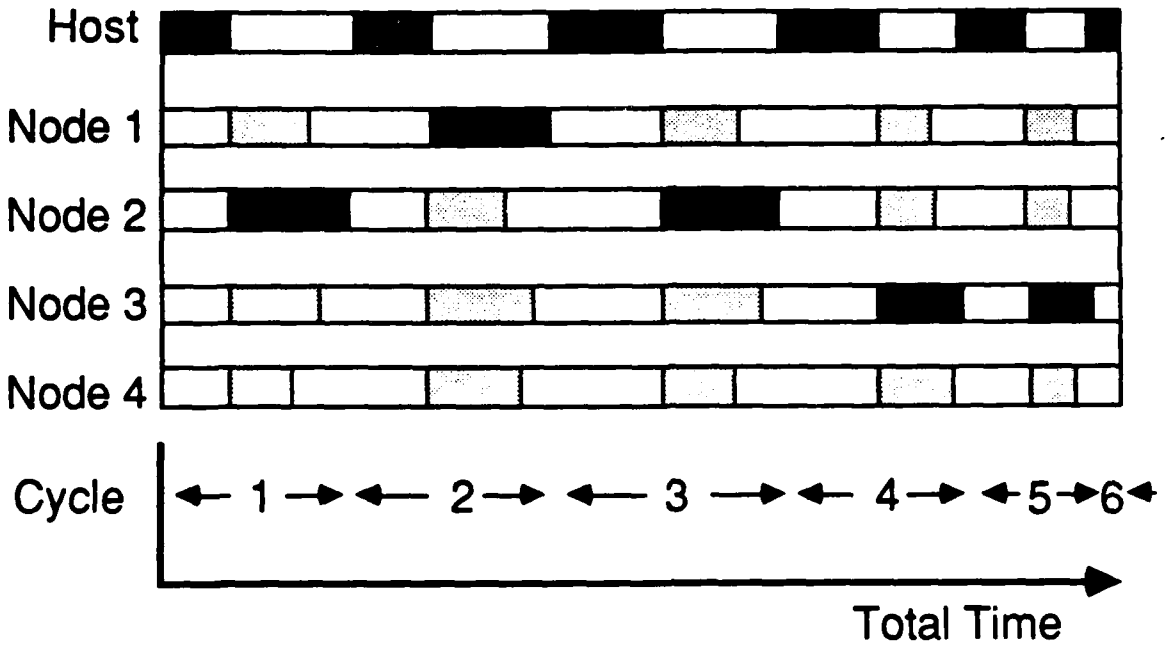

Figure 6 Typical Load-Balance in Standard Version of DECOMPAR

AD-A203214

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER MSP-87-4 R	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Decomposition of Linear Programs Using Parallel Computation		5. TYPE OF REPORT & PERIOD COVERED Revision of Technical Report MSP-87-4
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) James K. Ho Tak C. Lee R. P. Sundarraj		8. CONTRACT OR GRANT NUMBER(s) N0014-87-J-0163
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Tennessee, College of Bus. Admin. Department of Management 615 Stokely Management Center Knoxville, TN 37996		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Department of the Navy 800 N. Quincy Street Arlington, VA 22217-5000		12. REPORT DATE December 1988 13. NUMBER OF PAGES 19 Pages
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale, its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Linear Programming
Large-Scale Systems
Decomposition
Parallel Computing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper describes DECOMPAR: an implementation of the Dantzig-Wolfe decomposition algorithm for block-angular linear programs using parallel processing of the subproblems. The software is based on a robust experimental code for LP decomposition and runs on the CRYSTAL multi-computer at the University of Wisconsin-Madison. Initial computational experience is reported. Promising directions in future development of this approach are discussed.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)